

000110-01000

UNITED STATES PATENT APPLICATION FOR

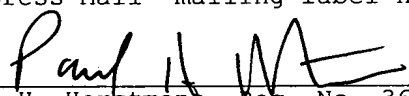
PERFORMANCE MONITORING IN DISTRIBUTED
SYSTEMS USING SYNCHRONIZED CLOCKS
AND DISTRIBUTED EVENT LOGS

Inventor:
Jefferson B. Burch

CERTIFICATE OF MAILING BY "EXPRESS MAIL"
UNDER 37 C.F.R. § 1.10

"Express Mail" mailing label number: EJ138444430US
Date of Mailing: 1-10-2000

I hereby certify that this correspondence is
being deposited with the United States Postal
Service, utilizing the "Express Mail Post Office to
Addressee" service addressed to **Assistant
Commissioner for Patents, Washington, D.C. 20231** and
mailed on the above Date of Mailing with the above
"Express Mail" mailing label number.



Paul H. Horstmann, Reg. No. 36,167
Signature Date: 1-10-2000

BACKGROUND OF THE INVENTION

Field of Invention

5 The present invention pertains to the field of distributed systems. More particularly, this invention relates to performance monitoring in distributed systems using synchronized clocks and distributed event logs.

10 Art Background

Distributed systems are commonly employed in a variety of applications. A typical distributed system includes a set of nodes that communicate via a communication network. One or more of the nodes
15 usually include processing resources that execute software for a distributed application. Examples of distributed applications include web client-server applications, groupware applications, and industrial and other types of control systems.

20 A distributed application may be viewed as a arrangement of software components that are distributed among the nodes of a distributed system. Examples of software components of a distributed
25 application include processes, file systems, database servers, web servers, client applications, server applications, network components, and sensor, actuator, and controller applications. Such software
30 components typically interact with one another using mechanisms such as function calls, messages, HTTP commands, etc., to name a few examples. An interaction between software components of a

distributed application may be referred to as an event.

5 Events that are generated in one location of a distributed application typically cause events to occur at other locations in the distributed application. In a web-based application, for example, an end-user may click a button in a web browser. The click typically generates events in the
10 form of HTTP commands. Each HTTP command in turn usually generates other events at other locations in the distributed application to communicate the HTTP command to a web server, for example via a TCP/IP link established by the protocol stacks in each node.
15 In response, a web server as the remote portion of the distributed application typically generates events such as SQL statements for database access or events for file system access to carry out the HTTP command as well as events to return the appropriate
20 information to the requesting web browser.

25 A capability to record the timing of events in a distributed application may be useful for a variety of system management tasks such as performance monitoring, diagnosis, and capacity planning. For example, a record of the timing of events may be useful for identifying bottlenecks in a distributed application that hinder overall performance.
30 Unfortunately, prior methods for performance monitoring usually record the timing of events in a single node. Prior methods typically do not provide event timing across multiple nodes of a distributed application.

SUMMARY OF THE INVENTION

5 A distributed system is disclosed that provides performance monitoring capability across multiple nodes of a distributed application. A distributed system according to the present techniques includes a set of nodes that communicate via a network. A distributed application is performed by a set of cooperating node applications executing in the nodes.

10 The distributed system implements techniques for generating time-stamp records for each of a set of significant events associated with one or more of the node applications. The time-stamp records provides a synchronized time base across the nodes for the

15 significant events. This enables temporal ordering of the significant events.

20 Other features and advantages of the present invention will be apparent from the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is described with respect to particular exemplary embodiments thereof and
5 reference is accordingly made to the drawings in which:

Figure 1 shows a distributed system that includes the performance monitoring techniques
10 disclosed herein;

Figure 2 illustrates the capture of time-stamp records for events associated with the node applications;
15

Figure 3 is a graph constructed from the captured time-stamp records with their synchronized time base;

Figure 4 shows a method for performance monitoring in a distributed application using the techniques disclosed herein;
20

Figure 5 shows a synchronized clock in one embodiment of a node.
25

0940041-01000

DETAILED DESCRIPTION

Figure 1 shows a distributed system 10 that includes the performance monitoring techniques disclosed herein. The distributed system 10 includes a set of nodes 20-24 that communicate via a network 30. Each of the nodes 20-24 include processor resources for executing a set of node applications 50-54, respectively.

The node applications 50-54 interact via the communication network 30 when performing a distributed application in the distributed system 10. The distributed application performed by the node applications 50-54 may be any type of distributed application such as web-based client-server applications including those used in e-commerce, groupware applications, order processing applications, or inventory management applications to name a few examples. In addition, the distributed application performed by the node applications 50-54 may be a control system application in which the nodes 20-24 include the appropriate control system hardware including sensors and actuators.

In one embodiment, each of the nodes 20-24 each include a corresponding synchronized clock 40-44. The synchronized clocks 40-44 maintain time values which are synchronized with respect to one another. These synchronized time values enable the coordination of event timing measurements throughout a distributed application in the distributed system 10.

5 The nodes 20-24 each implement a corresponding
event time-stamp recorder function 68, 78, and 88.
Each event time-stamp recorder function 68, 78, and
88 is call-able by the corresponding node application
50-54. Each event time-stamp recorder function 68,
78, and 88 when called obtains a time value from the
corresponding synchronized clock 40-44 and writes the
time value and an event code provided by the
corresponding node application 50-54 into a
10 corresponding event log 90-94 as a time-stamp record.

15 Applications in nodes that do not have
synchronized clocks can also be monitored. For
example, companion nodes may be associated with the
nodes that do not have synchronized clocks. Events
of interest on the nodes that do not have
synchronized clocks are time-stamped and recorded
using the corresponding companion nodes.
20 Communication paths to companion nodes can be a
physical wire, serial or parallel communication path,
etc. In one embodiment, event codes from an
application executing on a node that does not have a
synchronized clock is passed to the event time-stamp
recorder on its companion node over the communication
25 path to the companion node. Event logs may be
implemented on either node.

30 The information recorded in the event logs 90-94
may be read by any of the nodes 20-24 or other nodes
reachable via the network 30. This information may
be used to determine a variety of performance
indications for the distributed application performed

by the node applications 50-54. The information may be used to construct graphical displays that indicate timing associated with a set of predetermined events of interest in a distributed application.

5

In one embodiment, the time values in the synchronized clocks 40-44 are synchronized using a synchronization protocol described in U.S. Patent no. 5,566,180.

10

In another embodiment, the nodes 20-24 implement the network time protocol (NTP). In accordance with NTP, the nodes 20-24 periodically exchange messages via the network 30. Each message contains a time value from the synchronized clock 40-44 of the node 20-24 that originated the message. In response, each node 20-24 adjusts its corresponding synchronized clock 40-44. Eventually, the synchronized clocks 40-44 in the nodes 20-24 converge to synchronized time values.

15

20

The network 30 may be a packetized network such as Ethernet or a network such as LonTalk which is adapted to control systems. Alternatively, the network 30 may be implemented as a serial or parallel communication bus or other mechanism for communication.

25

Figure 2 illustrates the capture of time-stamp records for events associated with the node applications 50-52. The node application 50 includes

5 a set of functions 60-64 which are application-specific and a network stack 66 that performs communication via the network 30. Similarly, the node application 52 includes a set of functions 70-74 which are application-specific and a network stack 76 that performs communication via the network 30.

10 The particulars of the functions 60-64 and 70-74 depend on the particular distributed application performed by the node applications 50-52. For example, the functions 60-64 may be associated with a web browser application and the functions 70-74 may be associated with a web server application. The network stacks 66 and 76 bridge the underlying
15 physical network 30 to the node applications 50-52.

20 The application 50 is designed such that a call to the function 62 by the function 60 is deemed an event of significance in the execution of the corresponding distributed application. As a consequence, the function 60 calls the event time-stamp recorder function 68 near the time of the call to the function 62. The function 60 passes an event code=c1 parameter with the call to the event time-stamp recorder function 68. In response, the event
25 time-stamp recorder function 68 reads a time value (t1) from the synchronized clock 40 and writes the value-pair c1-t1 to an entry in the event log 90 as a time-stamp record.

5

15

25

In some embodiments, the function 70 may respond to the above sequence by calling the function 72, which calls the function 74, which calls the network stack 76 to transfer a message back to the node application 50 on up to the function 60. This provides an event loop which is originated by the node application 50 and completed by the node application 52 with the reply being returned to the node application 50. On the return path, different codes may be recorded in the event logs along the function call chain.

In an example embodiment, the node application 50 is a web browser and the node application 52 is a web server. The function 60 generates HTTP commands as events and time values for these events recorded by the event time-stamp recorder function 68. HTTP commands received by the node application 52 that require a database access cause the function 70, a database access function, to be called and time-stamp records for these events are recorded by the event time-stamp recorder function 78. Similarly, the event time-stamp recorder function 78 records when the function 70 completes a database access and the event time-stamp recorder function 68 records when the results are provided back to the function 60 to complete a web browser-web server transaction loop.

In some time critical cases, hardware in a node automatically records a time-stamp for a significant event, for example the time of data collection for a

sensor or time of arrival of a network message. This time-stamp can be passed to an event time-stamp recorder along with the desired code to be stored in an event log.

5

Tables 1 and 2 show examples of information recorded in the event logs 90 and 92, respectively, for the example sequence described above.

Table 1

10	Event Code	Time-stamp
	c1	t1
	c2	t2
	c2(ret)	t7
	c1(ret)	t8

15

Table 2

	Event Code	Time-stamp
	c3	t3
	c4	t4
20	c4(ret)	t5
	c3(ret)	t6

25

This sequence of entries written in to the event logs 90 and 92 may be viewed as corresponding to a series of states S1 through S8 in the distributed application performed by the node applications 50-52. The states S1 through S4 correspond to the event codes-value pairs C1-t1, C2-t2, C3-t3, and C4-t4, respectively. The states S5 through S8 correspond to

the event codes-value pairs C4(ret)-t5, C3(ret)-t6, C2(ret)-t7, and C1(ret)-t8, respectively.

Table 3 aggregates the information recorded in the event logs 90 and 92, respectively, for the example sequence described above.

State Number	Node	Event Code	Time-Stamp
S1	20	C1	T1
S2	20	C2	T2
S3	22	C3	T3
S4	22	C4	T4
S5	22	C4 (ret)	T5
S6	22	C3 (ret)	T6
S7	20	C2 (ret)	T7
S8	20	C1 (ret)	T8

Figure 3 is a graph constructed from the information obtained from the event logs 90-92. The states S1 through S8 are shown plotted against the corresponding time values t1 through t8. The time values t1 through t8 which were obtained from the synchronized clocks 40-42 provide a synchronized time base for analyzing the timing of the states S1 through S8 even though these states represent events that occur in the separate nodes 20 and 22.

An examination of the graphs shows that the highest latency in the states S1 through S8 occur between states S2 and S3, between states S4 and S5, and between states S6 and S7. The transitions

5

20.

25

functions 60-64 and 70-74 and the network stacks 66
and 76 may directly read the synchronized clocks 40-
44 and directly write entries in the event logs 90-94
or these functions may be performed in hardware
5 and/or on companion nodes.

At step 104, one or more experiments are run in
the distributed application to generate the
significant events and capture time-stamp records.
10 In the example above, HTTP commands are generated
using the node application 50 and are processed by
the node application 52, thereby generating
significant events which are recorded in the event
logs 90-92. Event loops such as the states S1
15 through S8 may be executed a large number of times to
gather time-stamp records.

At step 106, the time-stamp records captured
during step 104 are read and analyzed. In the
20 example above, the time-stamp records are read from
the event logs 90-94. Any computer system or node,
etc. having access to the network 30, including the
nodes 20-24, may read and analyze the time-stamp
records. Graphs may be generated such as the one in
25 **Figure 3** or other type of graph. For example, time-
stamp records from multiple loops through the states
S1 through S8 may be overlaid on a graph and/or may
be used to generate statistical distributions of time
values associated with transitions among the states
30 S1 through S8.

Figure 5 shows the synchronized clock 40 in one embodiment of the node 20. The synchronized clocks 42-44 may be implemented in a similar manner. The synchronized clock 40 includes a time packet recognizer 214, a clock 212, and a latch 210. The node 20 includes a physical interface 200 that enables transmission and reception of packets via the network 30. The physical interface 200 provides received packets to the time packet recognizer 214.

In this embodiment, the synchronized clock 40 maintains synchronized time in response to timing data packets and follow up packets which are transferred via the network 30. For example, a timing data packet 218 and a follow up packet 216 are carried on the network 30. The timing data packet 218 and the follow up packet 216 are generated by a master clock on the network 30. The master clock may be contained in one of the nodes 22-24 or on another node reachable via the network 30. The master clock may be a real-time clock.

The timing data packet 218 includes a delimiter 254 that identifies it as a timing data packet for the synchronization protocol of the synchronized clock 40. The follow up packet 216 includes a time stamp 250. The time stamp 250 indicates the local time in the master clock when the timing data packet 218 was generated.

Attorney Docket No. 10982344

The time packet recognizer 214 receives the timing data packet 218 through the physical interface 200. The time packet recognizer 214 detects a unique timing point in the recovered bit stream for the timing data packet 218. Upon detection of the unique timing point, the time packet recognizer 214 causes the latch 210 to latch a time value from the clock 212. The time value held in the latch 210 indicates the local time at which the time packet recognizer 214 received the timing data packet 218. Thereafter, the time packet recognizer 214 receives the follow up packet 216 and extracts the time stamp 250. The difference between the time stamp 250 and the time value in the latch 210 indicates the relative synchronization of the master clock and the clock 212. Once this difference is computed by the processor 202 it is used to adjust the time value in the clock 212 to conform it to the master clock.

The adjustment of the time value in the clock 212 may be accomplished by implementing the clock 212 as a counter driven by an oscillator with sufficient stability. The least significant few bits of the counter may be implemented as an adder so that an increment on oscillator periods may be occasionally increased or decreased to effectively speed up or slow down the clock 212 in accordance with the results of the computation of the difference between the time stamp 250 and the time held in the latch 210. The processor 202 when executing the event

time-stamp recorder function 68 reads the contents of the clock 212 to obtain time-stamps.

As a distributed application executes, the present teachings yield a set of time-stamp records in each node involved in the distributed application. Synchronized clocks in each node provide a synchronized time base that allows temporal ordering of the time-stamp records. This information is the basis for performance analysis of the end-to-end distributed application. In addition to summary statistical performance metrics (e.g. average end-to-end latency) details on individual distributed transactions can be captured with this system.

From the time-stamp records, an individual transaction can be analyzed to elucidate why it deviates from the norm. This detailed information provides key insights to software engineers and system programmers to understand the overall behavior of the distributed application.

Because this technique quickly identifies performance bottlenecks across the distributed system, engineering resources can be focused on the areas to optimize, tune, and/or redesign software/hardware. The resulting improvements can be monitored and analyzed with the present techniques to assess the real benefits of the changes and to focus engineering activities for the next round of improvements.

Through an iterative process of using the present techniques to identify performance bottlenecks and then focusing engineering resources to remove those bottlenecks, the performance of the distributed application can be subsequently improved.

An event time-stamp recorder can run in two modes including one in which a time-stamp and code are provided by application (usually by the hardware), and one in which a time-stamp is read by the event time-stamp recorder as it processes the code from the application.

In some embodiments, the event code of interest is presented to the application along with a time-stamp. In this embodiment, an event time-stamp recorder stores both the event code and time-stamp and does not read a local clock. For example, a measurement node in a distributed measurement and control system automatically generates a time-stamp when a measurement is made. If the application finds this measurement of interest, it passes the appropriate event code and time-stamp to an event time-stamp recorder for storage.

25

The present techniques provide the ability to analyze repetitions of a sequence of distributed transactions. Because time-stamp records from each repetition are captured, it is very useful to analyze how all the repetitions compare. For example, statistical descriptions of the complete transaction

such as the average delay for a transaction, the maximum delay, the minimum delay, and the statistical distribution (i.e. histogram).

5 A pair states may be selected and the delay between them analyzed. This provides a portion of the delay for the entire transaction and similar information such as average, max, min, and distribution may be calculated.

10

 The fact the present techniques yield a time-stamp for of every state across a distributed application provides more information in comparison to other measurement techniques. This makes it
15 possible to analyze the statistical distribution over and above a simple average.

 It may be useful is to examine what is happening on different nodes. One goal of distributed
20 applications is to allow all the nodes to perform useful work at the same time (e.g parallel execution). While the server is working, the client is also performing some useful activity. The events and time-stamps from the different nodes yielded by
25 the present techniques enable the generation of diagrams that describe how much parallel work is really occurring.

 In addition to providing time-stamps, the
30 synchronized clock mechanisms disclosed herein also provide detailed statistics on how tightly

5
10

15

20

25

30

5

10

15

20

25

30

where significant events are infrequent. The nodes 20-24 are configured to watch for a signification event and when it is detected all the other nodes 20-24 get notified.

5

It is preferable that event logs 90-94 are large enough to store all the interesting events over the fetch phase of the post process interval plus a safety margin to hold all the events that have recently come in. The safety margin is sized such that each node will have sufficient time to perform the determination of whether an event is interesting and to send out the lock down message. The transit time and processing time for the lock down message should also be considered.

The present techniques may introduce slight delays in the execution of the distributed application. This delay may be measured using the present techniques and the delays removed in the post processing phase. To perform this correction, an application would execute a correction loop such as the following;

```
25 For (I=0; I<10000; I++) {  
    EventLog(startCorrectionCode);  
    EventLog(endCorrectionCode);  
}
```

30 The time-stamps from these pairs of time-stamp records can be subtracted and then analyzed to obtain an understanding of the average delay, the maximum

delay, the minimum delay, and the statistical distribution of delays. These facts can be used to correct the post analysis results of the full distributed application.

5

Because these delays may vary as the distributed application runs, the application programmer may enter two back to back calls to the EventLog() function as illustrated above at some point in the application. At the expense of event log space, the difference in these two time-stamps may be used to verify the timing overhead of the present techniques. It is preferable that the EventLog() be efficient and its execution time be much shorter than the running distributed application.

10
15

The foregoing detailed description of the present invention is provided for the purposes of illustration and is not intended to be exhaustive or to limit the invention to the precise embodiment disclosed. Accordingly, the scope of the present invention is defined by the appended claims.

20